

A FAST DIRECT LEAST SQUARES ALGORITHM FOR HIERARCHICALLY BLOCK SEPARABLE MATRICES*

KENNETH L. HO[†] AND LESLIE GREENGARD[‡]

Abstract. We present a fast algorithm for linear least squares problems governed by hierarchically block separable matrices. Such matrices are generally dense but *data-sparse* and can describe many important operators including those derived from the Green's functions of non-oscillatory elliptic partial differential equations. The algorithm is based on a recursive skeletonization procedure that exposes this sparsity and solves the dense least squares problem as a much larger, equality-constrained, sparse one. It relies on a sparse QR factorization coupled with iterative weighted least squares methods. In essence, our scheme consists of a direct component, comprised of matrix compression and factorization, followed by an iterative component to enforce certain equality constraints. In practice, the iteration typically converges in no more than two steps and can therefore also be viewed as direct. We illustrate the performance of the method for both overdetermined and underdetermined systems, with an emphasis on radial basis function approximation and efficient updating and downdating.

Key words. fast algorithms, multilevel matrix compression, recursive skeletonization, sparse QR decomposition, weighted least squares, deferred correction, radial basis functions

AMS subject classifications. 65F05, 65F20, 65F50, 65Y15

1. Introduction. The method of least squares is a powerful technique for the approximate solution of overdetermined systems and is often used for data fitting and statistical inference in applied science and engineering. In this paper, we will primarily consider the linear least squares problem

$$(1.1) \quad \min_x \|Ax - b\|,$$

where $A \in \mathbb{C}^{M \times N}$ is dense and full-rank with $M \geq N$, $x \in \mathbb{C}^N$, $b \in \mathbb{C}^M$, and $\|\cdot\|$ is the Euclidean norm. Formally, the solution is given by

$$(1.2) \quad x = A^+b,$$

where A^+ is the Moore-Penrose pseudoinverse of A , and can be computed directly via the QR decomposition at a cost of $\mathcal{O}(MN^2)$ operations [8, 40]. This can be prohibitive when M and N are large. If A is structured so as to support fast multiplication, then iterative methods such as LSQR [43] or GMRES [37, 45] present an attractive alternative. However, such solvers still have several key disadvantages when compared with their direct counterparts:

(i) The convergence rate of an iterative solver can depend strongly on the conditioning of the system matrix, which, for least squares problems, can sometimes be very poor. In such cases, the number of iterations required, and hence the computational cost, can be far greater than expected (if the solver succeeds at all). Direct methods, by contrast, are robust in that their performance does not degrade with conditioning. Thus, they are often preferred in situations where reliability is critical.

*This work was supported in part by the National Science Foundation under awards DGE-0333389 and DMS-1203554, by the U.S. Department of Energy under contract DEFG0288ER25053, and by the Air Force Office of Scientific Research under NSSEFF Program Award FA9550-10-1-0180.

[†]Courant Institute of Mathematical Sciences and Program in Computational Biology, New York University, New York, NY (ho@courant.nyu.edu).

[‡]Courant Institute of Mathematical Sciences, New York University, New York, NY (greenard@courant.nyu.edu).

TABLE 1.1

Examples of radial kernels $\phi(r)$ admitting HBS representations: $H_0^{(1)}$, zeroth order Hankel function of the first kind; K_0 , zeroth order modified Bessel function of the second kind.

Type	Name	Kernel		Notes
		2D	3D	
Green's function	Laplace	$\log r$	$1/r$	k not too large
	Helmholtz	$H_0^{(1)}(kr)$	e^{ikr}/r	
	Yukawa	$K_0(kr)$	e^{-kr}/r	$n = 1, 2, 3, \dots$
	Polyharmonic	$r^{2n} \log r$	r^{2n-1}	
Radial basis function	Multiquadric	$\sqrt{r^2 + c^2}$		c not too large
	Inverse multiquadric	$1/\sqrt{r^2 + c^2}$		

(ii) Standard iterative schemes are inefficient for multiple right-hand sides. With direct solvers, on the other hand, following an expensive initial factorization, the subsequent cost for each solve is typically much cheaper (e.g., only $\mathcal{O}(MN)$ work to apply the pseudoinverse given precomputed QR factors). This is especially important in the context of updating and downdating as the least squares problem is modified by adding or deleting data, which can be viewed as low-rank updates of the original system matrix.

In this paper, we present a fast direct least squares solver for a class of structured dense matrices called hierarchically block separable (HBS) matrices. Such matrices were introduced by Gillman, Young, and Martinsson [24], and possess a nested low-rank property that enables highly efficient data-sparse representations. The HBS matrix structure is closely related to that of \mathcal{H} - and \mathcal{H}^2 -matrices [33, 34, 35] and hierarchically semiseparable (HSS) matrices [15, 16, 51], and can be considered a generalization of the matrix features utilized by multilevel summation algorithms like the fast multipole method (FMM) [28, 29]. Many linear operators are of HBS form, notably integral transforms with asymptotically smooth radial kernels. This includes those based on the Green's functions of non-oscillatory elliptic partial differential equations [9]. Some examples are shown in Table 1.1; we highlight, in particular, the Green's functions

$$\phi_{\Delta}(r) = \frac{1}{r}, \quad \phi_{\Delta^2}(r) = r,$$

for the Laplace and biharmonic equations, respectively, in 3D, and their regularizations, the inverse multiquadric and multiquadric kernels

$$\phi_{\text{IMQ}}(r) = \frac{1}{\sqrt{r^2 + c^2}}, \quad \phi_{\text{MQ}}(r) = \sqrt{r^2 + c^2},$$

respectively (for c not too large). The latter are well-known within the radial basis function (RBF) community and have been used to successfully model smooth surfaces [13, 36]. Also of note is the 2D biharmonic Green's function, the so-called thin plate spline

$$(1.3) \quad \phi_{\text{TPS}}(r) = r^2 \log r,$$

which minimizes a physical bending energy [22]. For an overview of RBFs, see [10, 44].

Previous work on HBS matrices exploited their structure to build fast direct solvers for the square $M = N$ case [24, 38, 42] (similar methods are available for other structured formats). Here, we extend the approach of [38] to the rectangular $M \geq$

TABLE 1.2

Asymptotic complexities for the least squares solver when applied to the operators in Table 1.1 on data embedded in a d -dimensional domain: M and N , matrix dimensions; $M \geq N$.

d	Precomputation	Solution
1	$\mathcal{O}(M + N)$	$\mathcal{O}(M + N)$
2	$\mathcal{O}(M + N^{3/2})$	$\mathcal{O}(M + N \log N)$
3	$\mathcal{O}(M + N^2)$	$\mathcal{O}(M + N^{4/3})$

N case. Our algorithm relies on the multilevel compression and sparsity-revealing embedding of [38], and recasts the (unconstrained) dense least squares problem (1.1) as a much larger, equality-constrained, sparse one. This is solved via a sparse QR factorization coupled with iterative weighted least squares methods. For the former, we use the SuiteSparseQR package [20] by Tim Davis, while for the latter, we employ the iteration of Barlow and Vemulapati [5], which has been shown to converge in no more than two steps for problems that are not too ill-conditioned. Thus, our solver can, in practice, be viewed as a direct method.

It is useful to divide our algorithm into two phases: a precomputation phase, comprising matrix compression and factorization, followed by an iterative solution phase using the precomputed QR factors. Clearly, for a given matrix, only the solution phase must be executed for each additional right-hand side. Table 1.2 lists asymptotic complexities for both phases when applied to the operators in Table 1.1 on data embedded in a d -dimensional domain for $d = 1, 2$, or 3 . Although the estimates generally worsen as d increases, the solver achieves optimal $\mathcal{O}(M + N)$ complexity for both phases in *any* dimension in the special case that the source (column) and target (row) data are separated (i.e., the domain and range of the continuous operator are disjoint). This may have applications, for example, in partial charge fitting in computational chemistry [6, 23].

Our methods can also generalize to underdetermined systems ($M < N$) when seeking the minimal norm solution in L^2 , i.e., the equality-constrained least squares problem

$$(1.4) \quad \min_{Ax=b} \|x\|,$$

provided that the solution, which is also given by (1.2), is not too ill-conditioned with respect to A .

Fast direct least squares algorithms have been developed in other structured matrix contexts as well, in particular within the \mathcal{H} - and HSS matrix frameworks using various structured orthogonal transformation schemes [7, 14, 21, 26]. Our approach, however, is quite different and explicitly leverages the sparse representation of HBS matrices and the associated sparse matrix technology (e.g., the state-of-the-art software package SuiteSparseQR). This has the possible advantage of producing an algorithm that is easier to implement, extend, and optimize. For related work on other structured matrices including those of Toeplitz, Hankel, and Cauchy type, see, for instance, [31, 39, 47, 52] and references therein.

The remainder of this paper is organized as follows. In the next two sections, we collect and review certain mathematical preliminaries on HBS matrices (section 2) and equality-constrained least squares problems (section 3). In section 4, we describe our fast direct algorithm for both overdetermined and underdetermined systems. Complexity estimates are given in section 5, while section 6 discusses efficient updating and downdating in the context of our direct solver. Numerical results are reported

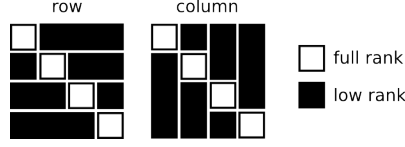


FIG. 2.1. A block separable matrix has low-rank off-diagonal block rows and columns (black); its diagonal blocks (white) can in general be full-rank.

in section 7. Finally, in section 8, we summarize our findings and end with some generalizations and concluding remarks.

2. HBS matrices. In this section, we define the HBS matrix property and discuss algorithms to compress such matrices and to sparsify linear systems governed by them. We will mainly follow the treatment of [38], extended to rectangular matrices in the natural way.

Let $A \in \mathbb{C}^{M \times N}$ be a matrix viewed with $p \times p$ blocks, with the i th row and column blocks having dimensions $m_i, n_i > 0$, respectively, for $i = 1, \dots, p$.

DEFINITION 2.1 (block separable matrix [24]). *The matrix A is block separable if each off-diagonal submatrix A_{ij} can be decomposed as the product of three low-rank matrices:*

$$(2.1) \quad A_{ij} = L_i S_{ij} R_j \quad \text{for } i \neq j,$$

where $L_i \in \mathbb{C}^{m_i \times k_i^r}$, $S_{ij} \in \mathbb{C}^{k_i^r \times k_j^c}$, and $R_j \in \mathbb{C}^{k_j^c \times n_j}$, with (ideally) $k_i^r \ll m_i$ and $k_j^c \ll n_j$, for $i, j = 1, \dots, p$.

DEFINITION 2.2 (off-diagonal block rows and columns). *The i th off-diagonal block row of A is the submatrix $[A_{i1} \cdots A_{i(i-1)} \ A_{i(i+1)} \cdots A_{ip}]$ consisting of the i th block row of A with the diagonal block A_{ii} removed; the off-diagonal block columns are defined analogously.*

Clearly, the block separability condition (2.1) is equivalent to requiring that the off-diagonal block rows and columns have low rank (Fig. 2.1). Observe that if A is block separable, then it can be written as

$$(2.2) \quad A = D + LSR,$$

where $D = \text{diag}(A_{ii})$, $L = \text{diag}(L_i)$, $R = \text{diag}(R_i)$, and $S = (S_{ij})$ is dense with $S_{ii} = 0$.

Let us now define a tree structure on the row and column indices $I = \{1, \dots, M\}$ and $J = \{1, \dots, N\}$, respectively, as follows. Associate with the root of the tree the entire index sets I and J . If a given subdivision criterion is satisfied (e.g., based on the sizes $|I|$ and $|J|$), partition the root node into a set of children, each associated with a subset of I and J such that they together span the whole sets. Repeat this process for each new node to be subdivided, partitioning its row and column indices among its children. In other words, if we label each tree node with an integer i and denote its row and column index sets by I_i and J_i , respectively, then

$$I_i = \bigcup_{j \in \text{child}(i)} I_j, \quad J_i = \bigcup_{j \in \text{child}(i)} J_j,$$

where $\text{child}(i)$ gives the set of node indices belonging to the children of node i . Furthermore, we also label the levels of the tree, starting with level 0 for the root at the coarsest level to level λ for the finest-level leaves; see Fig. 2.2 for an example.

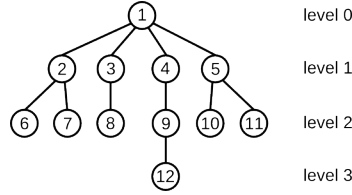


FIG. 2.2. An example of a tree on the row and column index sets with depth $\lambda = 3$. The root (node 1) contains all indices, which are hierarchically partitioned among its children.

Remark. Although we require that the number of row and column partitions be the same, we do not impose that $|I_i| = |J_i|$ for any node i . Indeed, it is possible for one of these sets to be empty.

Evidently, the tree defines a hierarchy among row and column index sets, each level of which specifies a block partition of the matrix A .

DEFINITION 2.3 (HBS matrix [24]). *The matrix A is HBS if it is block separable at each level of the tree hierarchy.*

HBS matrices arise in many applications, for example, when discretizing the kernels in Table 1.1 (up to a specified numerical precision), with row and column indices partitioned according to an octree-type ordering on the corresponding data [28, 29, 34, 38], which recursively groups together points that are geometrically collocated [46].

2.1. Multilevel matrix compression. We now review algorithms for computing the low-rank matrices in (2.1) characterizing the HBS form. Our primary tool for this task is the interpolative decomposition (ID) [17].

DEFINITION 2.4 (ID). *An ID of a matrix $A \in \mathbb{C}^{m \times n}$ with rank k is a factorization $A = BP$, where $B \in \mathbb{C}^{m \times k}$ consists of a subset of the columns of A and $P \in \mathbb{C}^{k \times n}$ contains the $k \times k$ identity matrix. We call B and P the skeleton and interpolation matrices, respectively.*

As stated, the ID clearly compresses the column space of A , but we can just as well compress the row space by applying the ID to A^T . Efficient algorithms for adaptively computing an ID to any specified precision are available [17, 41, 50].

DEFINITION 2.5 (row and column skeletons). *The row indices corresponding to the retained rows in the ID are called the row or incoming skeletons; the column indices corresponding to the retained columns are called the column or outgoing skeletons.*

A multilevel algorithm for the compression of HBS matrices follows naturally. For simplicity, we describe the procedure for matrices with a uniform tree depth (i.e., all leaves are at level λ), with the understanding that it extends easily to the adaptive case. The following scheme is known as recursive skeletonization (Fig. 2.3):

1. Starting at the leaves of the tree, extract the diagonal blocks and compress the off-diagonal block rows and columns using the ID to a specified precision $\epsilon > 0$ as follows. For each block $i = 1, \dots, p$, compress the row space of the i th off-diagonal block row and call L_i the corresponding row interpolation matrix. Similarly, for each block j , compress the column space of the j th off-diagonal block column and call R_j the corresponding column interpolation matrix. Let S be the skeleton submatrix of A , with each off-diagonal block S_{ij} for $i \neq j$ defined by the row and column skeletons associated with L_i and R_j , respectively.

2. Since the off-diagonal blocks S_{ij} are submatrices of the corresponding A_{ij} , the compressed matrix S is HBS and so can itself be compressed in the same way.

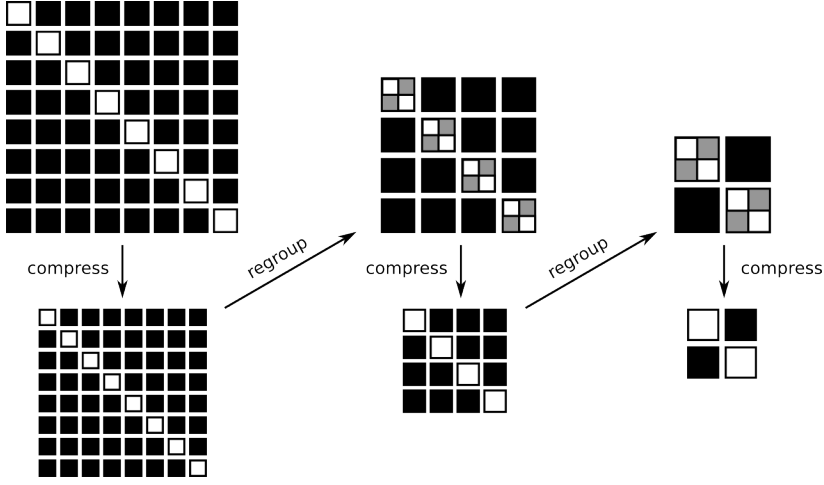


FIG. 2.3. Schematic of recursive skeletonization, comprising alternating steps of compression and regrouping via ascension of the index tree. The diagonal blocks (white and gray) are extracted at each level; they are shown here only to illustrate the regrouping process.

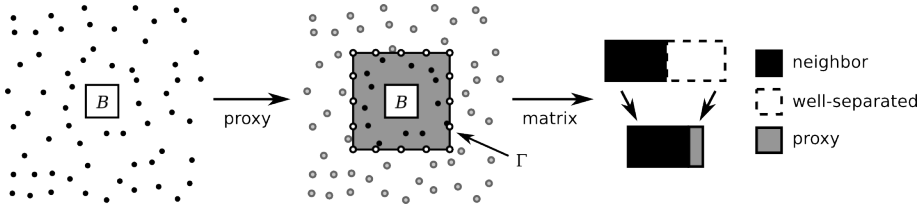


FIG. 2.4. The interaction of a region B with a distribution of exterior sources or targets (left) can be decomposed into neighboring and well-separated contributions. By representing the latter via a proxy surface Γ (center), the matrix dimension to compress against for the block row or column corresponding to B (right) can be reduced to just the number of points in the neighboring region plus a constant set of points on Γ .

Thus, move up one level in the tree, regroup the matrix blocks accordingly, and repeat.

The result is a telescoping matrix representation of the form

$$(2.3) \quad A \approx D^{(\lambda)} + L^{(\lambda)} \left[\dots D^{(2)} + L^{(2)} \left(D^{(1)} + L^{(1)} D^{(0)} R^{(1)} \right) R^{(2)} \dots \right] R^{(\lambda)},$$

cf. (2.2), where the superscript indexes the tree level $l = 0, \dots, \lambda$, that is accurate to precision approximately ϵ . The algorithm is automatically adaptive in the sense that the compression is more efficient if lower precision is required [24, 38].

Remark. For the kernels in Table 1.1, which obey some form of Green's theorem (at least approximately), it is possible to substantially accelerate the preceding algorithm by using a proxy surface to capture all far-field interactions (Fig. 2.4). The key idea is that any such interaction can be represented in terms of some equivalent density on an appropriate local bounding surface, which can be chosen so that it requires only a constant number of points to discretize, irrespective of the actual number of points in the far field or their detailed structure. This observation hence replaces each global compression step with an entirely local one; see [17, 27, 38, 42, 53] for details.

2.2. Structured sparse embedding. For $M = N$, the decomposition (2.3) enables a highly structured sparse representation of the linear system $Ax = b$ as

$$(2.4) \quad \begin{bmatrix} D^{(\lambda)} & L^{(\lambda)} & & & & & \\ R^{(\lambda)} & & -I & & & & \\ & -I & D^{(\lambda-1)} & \ddots & & & \\ & & \ddots & \ddots & -I & & \\ & & & -I & D^{(1)} & L^{(1)} & \\ & & & & R^{(1)} & & -I \\ & & & & & -I & D^{(0)} \end{bmatrix} \begin{bmatrix} x^{(\lambda)} \\ y^{(\lambda)} \\ \vdots \\ \vdots \\ x^{(1)} \\ y^{(1)} \\ x^{(0)} \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

under the identifications

$$(2.5a) \quad x^{(\lambda)} = x, \quad x^{(l)} = R^{(l+1)}x^{(l+1)} \quad \text{for } l = \lambda - 1, \dots, 0,$$

$$(2.5b) \quad y^{(1)} = D^{(0)}x^{(0)}, \quad y^{(l+1)} = D^{(l)}x^{(l)} + L^{(l)}y^{(l)} \quad \text{for } l = 1, \dots, \lambda - 1.$$

This expanded embedding clearly exposes the sparsity of HBS matrix equations and permits the immediate application of existing fast sparse solvers (such as UMFPACK [19] as in [38]).

If $M \geq N$, however, then we have to deal with the overdetermined problem (1.1), and (2.4) must be interpreted somewhat more carefully. In particular, the identities (2.5) still hold, so only the first block row of (2.4) is to be solved in the least squares sense. Thus, denoting the first block row of the sparse matrix in (2.4) by \mathbf{E} and the remainder (i.e., its last 2λ block rows) by \mathbf{C} , and defining $\mathbf{x} = (x^{(\lambda)}, y^{(\lambda)}, \dots, x^{(1)}, y^{(1)}, x^{(0)})^T$, the analogue of (2.4) for (1.1) is the *equality-constrained* least squares problem

$$(2.6) \quad \min_{\mathbf{C}\mathbf{x}=\mathbf{0}} \|\mathbf{E}\mathbf{x} - b\|,$$

where both \mathbf{E} and \mathbf{C} are sparse. It is easy to see that \mathbf{C} has full row rank.

Similarly, if $M < N$ and we seek to solve the underdetermined system (1.4), then the corresponding problem is

$$(2.7) \quad \min_{\mathbf{M}\mathbf{x}=\mathbf{b}_1} \|\mathbf{I}_1\mathbf{x}\|,$$

where

$$\mathbf{M} = \begin{bmatrix} \mathbf{E} \\ \mathbf{C} \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} b \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{I}_1 = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix},$$

i.e., \mathbf{M} is the entire sparse matrix on the left-hand side of (2.4), which also has full row rank; \mathbf{b}_1 is the right-hand side of (2.4); and \mathbf{I}_1 is an operator that picks out the first block row of the vector on which it acts.

3. Equality-constrained least squares. We now turn to the solution of linear least squares problems with linear equality constraints, with special attention to the case that both governing matrices are sparse as in (2.6) and (2.7). For consistency with the linear algebra literature, we adopt the notation of Barlow et al. [3, 4, 5],

which unfortunately conflicts somewhat with our previous definitions; the following notation is thus meant to pertain only to this section.

Hence, consider the problem

$$(3.1) \quad \min_{Cx=g} \|Ex - f\|,$$

where $E \in \mathbb{C}^{m \times n}$ and $C \in \mathbb{C}^{p \times n}$, with

$$\text{rank}(C) = p, \quad \text{rank} \left(\begin{bmatrix} E \\ C \end{bmatrix} \right) = n$$

so that the solution is unique. Classical reduction schemes for solving (3.1), such as the direct elimination and nullspace methods, require matrix products that can destroy the sparsity of the resulting reduced, unconstrained system [8, 40].

3.1. Weighted least squares. An attractive alternative when both E and C are sparse is the method of weighting, which recasts (3.1) in the unconstrained form

$$(3.2) \quad \min_x \|A(\tau)x - b(\tau)\|,$$

where

$$A(\tau) = \begin{bmatrix} E \\ \tau C \end{bmatrix}, \quad b(\tau) = \begin{bmatrix} f \\ \tau g \end{bmatrix}$$

for τ a suitably large weight. Clearly, as $\tau \rightarrow \infty$, the solution of (3.2) approaches that of (3.1). The advantage, of course, is that (3.2) can be solved using standard sparse techniques; this point of view is elaborated in [4, 48].

However, the choice of an appropriate weight can be a delicate matter: if τ is too small, then (3.2) approximates (3.1) poorly, while if τ is too large, then (3.2) can become ill-conditioned. An intuitive approach is to start with a small weight, then carry out some type of iterative refinement, effectively increasing the weight with each step. Such a scheme was first proposed by Van Loan [48], then further studied and improved by Barlow et al. [3, 5]; we summarize their results in the next section.

3.2. Iterative reweighting by deferred correction. In [5], Barlow and Vemulapati presented the following deferred correction procedure for the solution of the equality-constrained least squares problem (3.1) via the successive solution of the weighted problem (3.2) with a *fixed* weight τ :

1. Find

$$x^{(0)} = \arg \min_x \|A(\tau)x - b(\tau)\|$$

and set

$$r^{(0)} = f - Ex^{(0)}, \quad w^{(0)} = g - Cx^{(0)}, \quad \lambda^{(0)} = \tau^2 w^{(0)}.$$

2. For $k = 0, 1, 2, \dots$ until convergence, find

$$\Delta x^{(k)} = \arg \min_x \|A(\tau)x - b^{(k)}(\tau)\|, \quad b^{(k)}(\tau) = \begin{bmatrix} \tau w^{(k)} + \tau^{-1} \lambda^{(k)} \\ r^{(k)} \end{bmatrix}$$

and update

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \Delta x^{(k)}, \\ r^{(k+1)} &= r^{(k)} - E \Delta x^{(k)}, \\ w^{(k+1)} &= w^{(k)} - C \Delta x^{(k)}, \\ \lambda^{(k+1)} &= \lambda^{(k)} + \tau^2 w^{(k+1)}. \end{aligned}$$

Terminate when the constraint residual $\|w^{(k+1)}\|$ is small.

Since τ is fixed, a single precomputed QR factorization of $A(\tau)$ can be used for all iterations. This algorithm is a slight modification of that employed by Van Loan [48] and has been shown to converge to the correct solution for τ appropriately chosen, provided that (3.1) is not too ill-conditioned [3, 5, 48] (see also section 4.3). In particular, if implemented in double precision, then for $\tau = \epsilon_{\text{mach}}^{-1/3} \sim 1.7 \times 10^5$, where ϵ_{mach} is the machine epsilon, Barlow and Vemulapati [5] showed that their algorithm requires no more than two iterations to produce an accurate result. Thus, for a broad class of problems of the form (3.1) for which it is reasonable to expect an accurate answer, the above scheme can be considered a *direct* method, which can be made explicit by running the iteration for exactly two steps.

Remark. Although Barlow and Vemulapati [5] considered (3.1) only over the reals, there is no inherent difficulty in extending their solution procedure to the complex case.

4. Algorithm. We are now in a position to describe our fast direct method for solving the over- and underdetermined systems (1.1) and (1.4), respectively, when A is HBS. Let $\epsilon > 0$ be a specified compression precision and set $\tau = \epsilon_{\text{mach}}^{-1/3} \sim 1.7 \times 10^5$; we assume that all calculations are performed in double precision.

4.1. Overdetermined systems. Let $A \in \mathbb{C}^{M \times N}$ be HBS with $M \geq N$. Our algorithm proceeds in two phases. First, for the precomputation phase:

1. Compress A to precision ϵ using recursive skeletonization [38].
2. Compute a sparse QR factorization of the weighted sparse matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{E} \\ \tau \mathbf{C} \end{bmatrix},$$

where \mathbf{E} and \mathbf{C} are as defined in section 2.2.

This is followed by the solution phase, which, for a given right-hand side $b \in \mathbb{C}^M$, produces an approximate solution (1.2) by using the precomputed QR factors to solve the equality-constrained least squares embedding (2.6) via deferred correction [5]. Clearly, for a fixed matrix A , only the solution phase must be performed for each additional right-hand side. Therefore, the cost of the precomputation phase is amortized over all such solves.

Remark. The algorithm can also easily accommodate various modifications of the standard system (1.1), e.g., the problem

$$(4.1) \quad \min_x \|Ax - b\|^2 + \mu \|x\|^2$$

with Tikhonov regularization, which can be solved using

$$\mathbf{A} = \begin{bmatrix} \mathbf{E} \\ \mu \mathbf{I}_1 \\ \tau \mathbf{C} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$

in the weighted formulation (3.2).

4.2. Underdetermined systems. Now let $A \in \mathbb{C}^{M \times N}$ be HBS with $M < N$. Then (1.4) can be solved using the same algorithm as above but with

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_1 \\ \tau \mathbf{M} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ \mathbf{b}_1 \end{bmatrix}.$$

4.3. Error analysis. Let $\kappa(A) = \|A\| \|A^+\|$ be the condition number of A . Then provided that $\epsilon \kappa(A) \ll 1$, the exact solution of the least squares problem (2.6) or (2.7) governed by the compressed matrix A_ϵ in (2.3) is accurate to precision $\mathcal{O}(\epsilon \kappa(A))$ with respect to the true solution (1.2) (see [38]). Furthermore, standard estimates show that $\kappa(A_\epsilon) = \mathcal{O}(\kappa(A))$. Therefore, if A is not too ill-conditioned, then neither is A_ϵ , so the deferred correction procedure will succeed. Thus, we may expect a relative error of the order $\mathcal{O}(\epsilon \kappa(A))$.

5. Complexity analysis. In this section, we analyze the complexity of our solver for a representative example: the HBS matrix A defined by a kernel from Table 1.1, acting on source and target data distributed uniformly over the same d -dimensional domain (but at different densities). We follow the approach of [38].

Sort both sets of data together in one hyperoctree (the multidimensional generalization of an octree, cf. [46]) as outlined in section 2, subdividing each node until it contains no more than a set number of combined row and column indices, i.e., $|I_i| + |J_i| = \mathcal{O}(1)$ for each leaf i . For each tree level $l = 0, \dots, \lambda$, let p_l denote the number of matrix blocks; m_l and n_l , the row and column block sizes, respectively, in the compressed representation (2.3), assumed equal across all blocks for simplicity; and k_l , the skeleton block size, which is clearly of the same order for both rows and columns as it depends only on $\min\{\mathcal{O}(m_l), \mathcal{O}(n_l)\}$ (this can be made obvious by explicitly considering proxy compression, which produces interaction matrices of dimension $\mathcal{O}(m_l) \times \mathcal{O}(n_l)$). Note that m_l and n_l are *not* the row and column block sizes in the tree; they are the result of hierarchically “pulling up” skeletons during the compression process (see (iv) below). Moreover, since $M \neq N$ in general, we can define an additional level parameter $\lambda' \leq \lambda$ corresponding to the depth of the tree constructed via the same process on only the smaller of the source or target data, e.g., on only the source data if $M \geq N$. For the remainder of this discussion, we assume that $M \geq N$. Analogous results can be recovered for $M < N$ simply by switching the roles of M and N in what follows. We start with some useful observations:

(i) By construction, $p_\lambda(m_\lambda + n_\lambda) = M + N \sim M$, where $m_\lambda, n_\lambda = \mathcal{O}(1)$, so $p_\lambda \sim M$. Similarly, $p_{\lambda'} \sim N$.

(ii) Each subdivision increases the number of blocks by a factor of roughly 2^d , so $p_{l+1} \sim 2^d p_l$. In particular, $p_0 = 1$, so $\lambda \sim (1/d) \log M$ and $\lambda' \sim (1/d) \log N$.

(iii) For $l = \lambda' + 1, \dots, \lambda$, $k_l = \mathcal{O}(1)$ since $\min\{\mathcal{O}(m_l), \mathcal{O}(n_l)\} = \mathcal{O}(1)$, while for $l = 0, \dots, \lambda'$, it can be shown [38] that

$$(5.1) \quad k_l \sim \begin{cases} l & \text{if } d = 1, \\ 2^{(d-1)l} & \text{if } d > 1. \end{cases}$$

(iv) The total number of row and column indices at level $l < \lambda$ is equal to the total number of skeletons at level $l + 1$, i.e., $p_l m_l = p_l n_l = p_{l+1} k_{l+1}$, so $m_l, n_l \sim k_{l+1}$.

5.1. Matrix compression. From [17, 41, 50], the cost of computing a rank- k ID of an $m \times n$ matrix is $\mathcal{O}(kmn)$. If proxy compression is used, then $m \sim m_l$ for a

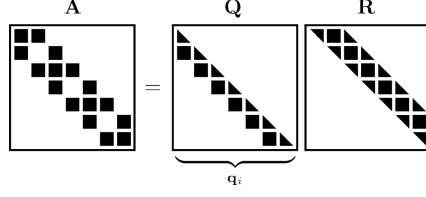


FIG. 5.1. Sparsity structure of QR factors for the structured embedding (2.4) with $M = N$, where the orthogonal matrix \mathbf{Q} is given in terms of the elementary Householder reflectors \mathbf{q}_i .

block at level l . Therefore, the total cost of matrix compression is

$$T_{\text{cm}} \sim \sum_{l=0}^{\lambda} p_l k_l m_l n_l \sim \sum_{l=0}^{\lambda} p_l k_l^3.$$

We break this into two sums, one over $l = \lambda' + 1, \dots, \lambda$ and another over $l = 0, \dots, \lambda'$, with estimates

$$\sum_{l=\lambda'+1}^{\lambda} p_l k_l^3 \sim M, \quad \sum_{l=0}^{\lambda'} p_l k_l^3 \sim \begin{cases} N & \text{if } d = 1, \\ N^{3(1-1/d)} & \text{if } d > 1, \end{cases}$$

respectively. Hence,

$$(5.2) \quad T_{\text{cm}} \sim \begin{cases} M + N & \text{if } d = 1, \\ M + N^{3(1-1/d)} & \text{if } d > 1. \end{cases}$$

5.2. Compressed QR factorization. We consider the QR decomposition of a block tridiagonal matrix with the same sparsity structure as that of \mathbf{M} in (2.4), computed using Householder reflections. This clearly encompasses the factorization of $\mathbf{A} = \mathbf{Q}\mathbf{R}$ for both over- and underdetermined systems. We begin by studying the square $M = N$ case, for which it is immediate that \mathbf{R} is block upper bidiagonal (Fig. 5.1), so the cost of Householder triangularization is

$$(5.3) \quad T_{\text{qr}} \sim \sum_{l=0}^{\lambda} p_l m_l n_l^2 \sim \sum_{l=0}^{\lambda} p_l k_l^3 \sim \begin{cases} M + N & \text{if } d = 1, \\ M + N^{3(1-1/d)} & \text{if } d > 1, \end{cases}$$

following the structure of \mathbf{R} . The $M > N$ and $M < N$ cases are easily analyzed by noting that only the blocks at level λ are rectangular, so any nonzero propagation during triangularization is limited and both essentially reduce to the square case above (Fig. 5.2).

The complexities (5.2) and (5.3) of compression and factorization, respectively, together constitute the cost of the precomputation phase.

5.3. Compressed pseudoinverse application. We now examine the cost of applying the pseudoinverse to solve the weighted least squares problem (3.2) using the precomputed QR factors. For this, we suppose that the solution is determined via the equation $\mathbf{R}\mathbf{x} = \mathbf{Q}^*\mathbf{b}$, which requires one application of \mathbf{Q}^* , assumed to be performed using elementary Householder transformations, and one backsolve with \mathbf{R} , whose cost is clearly on the same order as multiplying by \mathbf{R} . Then from the arguments above, it

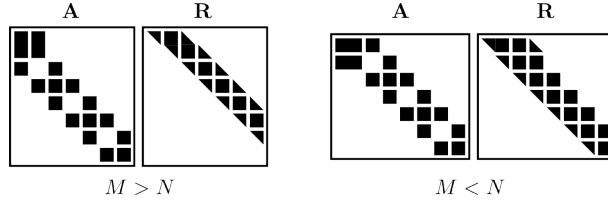


FIG. 5.2. Sparsity structure of the triangular QR factor \mathbf{R} for the structured embedding (2.4) in the $M > N$ (left) and $M < N$ (right) cases.

is evident that both operations have complexity

$$T_{\text{sv}} \sim \sum_{l=0}^{\lambda} p_l m_l n_l \sim \sum_{l=0}^{\lambda} p_l k_l^2 \sim \begin{cases} M + N & \text{if } d = 1, \\ M + N \log N & \text{if } d = 2, \\ M + N^{2(1-1/d)} & \text{if } d > 2. \end{cases}$$

Since the total number of such problems to be solved is constant for each outer problem (2.6), cf. section 3.2, this is also the complexity of the solution phase.

Remark. One can also use the seminormal equations $\mathbf{R}^* \mathbf{R} \mathbf{x} = \mathbf{A}^* \mathbf{b}$, which do not require the orthogonal matrix \mathbf{Q} [8]. However, one step of iterative refinement is necessary for stability, so the total cost is three applications of \mathbf{A} or \mathbf{A}^* (one each for the original and refinement solves, plus another to compute the residual), and four solves with \mathbf{R} or \mathbf{R}^* . In practice, we found this approach to be slower than that involving \mathbf{Q} by a factor of about four.

5.4. Some remarks. For all complexities above, the constants implicit in the estimates are of the form $\mathcal{O}(2^d \log^\alpha \epsilon)$ for modest α , i.e., they are exponential in the dimension and polylogarithmic in the precision [28, 29].

In the special case that the source and target data are separated, $k_l = \mathcal{O}(1)$ for all l , so T_{cm} , T_{qr} , and T_{sv} all have optimal complexity $\mathcal{O}(M + N)$ in *any* dimension. This describes, for example, the fitting of atomic partial charges to reproduce electrostatic potential values on “shells” around a molecule [6, 23], the computation of equivalent densities in the kernel-independent FMM [53], and even the calculation of the ID in recursive skeletonization [38], which requires a least squares solve [17].

6. Updating and downdating. We now discuss an important feature of our direct solver: its capacity for efficient updating and downdating in response to dynamically changing data. Our methods are based on the augmented system approach of [27], extended to the least squares setting, and exploit the ability to rapidly apply A^+ via the solution phase of our algorithm, which we hereafter take as a computational primitive. Thus, suppose that we are given some base linear system (1.1), focusing for simplicity on the overdetermined case, for which we have precomputed a compressed QR factorization of A . We consider the addition and deletion of both rows and columns, corresponding to the modification of observations and regression variables, respectively. Furthermore, we assume that such modifications are small, in particular so that the system remains overdetermined, and accommodate each case within the framework of the general equality-constrained least squares problem

$$(6.1) \quad \min_{\mathbf{C}\mathbf{x}=\mathbf{g}} \|\mathbf{E}\mathbf{x} - \mathbf{f}\|.$$

6.1. Adding and deleting rows. To add p_r rows to the matrix $A = (a_{ij})$, and correspondingly to the vector $b = (b_1, \dots, b_M)^\top$, in (1.1), we simply use (6.1) with

$$(6.2) \quad \mathbf{E} = \begin{bmatrix} A \\ C_+^r \end{bmatrix}, \quad \mathbf{C} = 0, \quad \mathbf{x} = x, \quad \mathbf{f} = \begin{bmatrix} b \\ b_+ \end{bmatrix}, \quad \mathbf{g} = 0,$$

where $C_+^r \in \mathbb{C}^{p_r \times N}$ describes the influence of the variables x on the new data b_+ . To delete q_r rows with indices k_1, \dots, k_{q_r} , we add q_r degrees of freedom to those rows to be deleted in order to enforce strict agreement with those observations as follows:

$$\mathbf{E} = \begin{bmatrix} A & B_-^r \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C_-^r & I \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ x_-^r \end{bmatrix}, \quad \mathbf{f} = b, \quad \mathbf{g} = d,$$

where $B_-^r = (\delta_{k_{ij}}) \in \mathbb{C}^{M \times q_r}$, $C_-^r = (a_{k_{ij}}) \in \mathbb{C}^{q_r \times N}$, and $d = (b_{k_1}, \dots, b_{k_{q_r}})^\top$; here,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j \end{cases}$$

is the Kronecker delta. The simultaneous addition and deletion of rows can be achieved via a straightforward combination of the above:

$$\mathbf{E} = \begin{bmatrix} A & B_-^r \\ C_+^r & \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C_-^r & I \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ x_-^r \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} b \\ b_+ \end{bmatrix}, \quad \mathbf{g} = d.$$

6.2. Adding and deleting columns. To add p_c columns to A and hence to the vector $x = (x_1, \dots, x_N)^\top$, we let

$$\mathbf{E} = \begin{bmatrix} A & B_+^c \end{bmatrix}, \quad \mathbf{C} = 0, \quad \mathbf{x} = \begin{bmatrix} x \\ x_+^c \end{bmatrix}, \quad \mathbf{f} = b, \quad \mathbf{g} = 0,$$

where $B_+^c \in \mathbb{C}^{M \times p_c}$ describes the influence of the new variables x_+^c on the data. To delete q_c columns with indices l_1, \dots, l_{q_c} , we add q_c “anti-variables” annihilating their effects:

$$\mathbf{E} = \begin{bmatrix} A & B_-^c \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C_-^c & I \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ x_-^c \end{bmatrix}, \quad \mathbf{f} = b, \quad \mathbf{g} = 0,$$

where $B_-^c = (a_{il_j}) \in \mathbb{C}^{M \times q_c}$ and $C_-^c = (\delta_{il_j}) \in \mathbb{C}^{q_c \times N}$. Finally, to add and delete columns simultaneously, we take

$$\mathbf{E} = \begin{bmatrix} A & B_+^c & B_-^c \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C_-^c & 0 & I \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ x_+^c \\ x_-^c \end{bmatrix}, \quad \mathbf{f} = b, \quad \mathbf{g} = 0.$$

6.3. Simultaneous modification of rows and columns. The general case of modifying both rows and columns can thus be treated using (6.1) with

$$\mathbf{E} = \begin{bmatrix} A & B_+^c & B_-^r & B_-^c \\ C_+^r & D_+ & 0 & D_- \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C_-^r & 0 & I & 0 \\ C_-^c & 0 & 0 & I \end{bmatrix}$$

and

$$\mathbf{x} = \begin{bmatrix} x \\ x_+^c \\ x_-^r \\ x_-^c \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} b \\ b_+ \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} d \\ 0 \end{bmatrix},$$

where $D_+ \in \mathbb{C}^{p_r \times p_c}$ describes the influence of x_+^c on b_+ , $D_- = (c_{il_j}) \in \mathbb{C}^{p_r \times q_c}$ for $C_+^r = (c_{ij})$ accounts for the effect of column deletion on b_+ (alternatively, one can zero out the relevant columns in C_+^r), and all other quantities are as defined previously.

6.4. Solution methods. The augmented system (6.1) can be solved by deferred correction [5], where the matrix to be considered at each step is

$$\mathbf{A} = \begin{bmatrix} \mathbf{E} \\ \tau \mathbf{C} \end{bmatrix} \equiv \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

for $B \in \mathbb{C}^{M \times n}$, $C \in \mathbb{C}^{m \times N}$, and $D \in \mathbb{C}^{m \times n}$, where $m = p_r + q_r + q_c$ and $n = p_c + q_r + q_c$. If m and n are small, then an efficient approach is to compute \mathbf{A}^+ by using A^+ in various block pseudoinverse formulas (see, e.g., [11]) or by invoking Greville's method [12, 30], which can construct \mathbf{A}^+ via a sequence of rank-one updates to A^+ . Alternatively, one can appeal to iterative methods like GMRES [45], using A^+ as a preconditioner. In this approach, instead of solving

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|, \quad \mathbf{b} = \begin{bmatrix} \mathbf{f} \\ \tau \mathbf{g} \end{bmatrix},$$

we consider instead, say, the left preconditioned system

$$\min_{\mathbf{x}} \|\mathbf{B}\mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{b}\|$$

for an appropriate choice of the preconditioner $\mathbf{B} \in \mathbb{C}^{(N+n) \times (M+m)}$. Hayami, Yin, and Ito [37] showed that GMRES converges provided that $\text{range}(\mathbf{A}) = \text{range}(\mathbf{B}^*)$ and $\text{range}(\mathbf{A}^*) = \text{range}(\mathbf{B})$. Therefore, suitable choices of \mathbf{B} include, e.g.,

$$\begin{bmatrix} A^+ & C^* \\ B^* & D^* \end{bmatrix}, \quad \begin{bmatrix} A^+ & C^* \\ B^* & D^+ \end{bmatrix}$$

(the latter if D is not rank-deficient), for which one application of A^+ is required per iteration. Such methods also have the possible advantage of being more flexible and robust.

7. Numerical results. In this section, we report some numerical results for our fast direct solver, compared primarily against LAPACK/ATLAS [1, 49]. We considered problems in both 2D and 3D. All matrices were block partitioned using quadrees in 2D and octrees in 3D, uniformly subdivided until all leaf nodes contained no more than a fixed number of combined rows and columns (cf. sections 2.1 and 5), while adaptively pruning all empty nodes during the refinement process. The recursive skeletonization algorithm was implemented in Fortran and employed as described in [38]. Sparse QR factorizations were computed with SuiteSparseQR [20] through a MATLAB R2012b (The MathWorks, Inc.: Natick, MA) interface, keeping all orthogonal matrices in compact Householder form. The deferred correction procedure [5] was implemented in MATLAB. All calculations were performed in double-precision real arithmetic on a single 3.10 GHz processor with 4 GB of RAM. In each case, we checked our results against those produced by either LAPACK/ATLAS or a GMRES-based iterative solver.

7.1. Laplace's equation. For benchmarking purposes, we first applied our method to Laplace's equation

$$\Delta u = 0 \quad \text{in } \Omega \in \mathbb{R}^d, \quad u = f \quad \text{on } \partial\Omega$$

in a simply connected interior domain with Dirichlet boundary conditions, which can be solved by writing the solution in the form of a double-layer potential

$$u(\vec{x}) = \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\vec{y}}}(\|\vec{x} - \vec{y}\|) \sigma(\vec{y}) d\vec{y} \quad \text{for } \vec{x} \in \Omega,$$

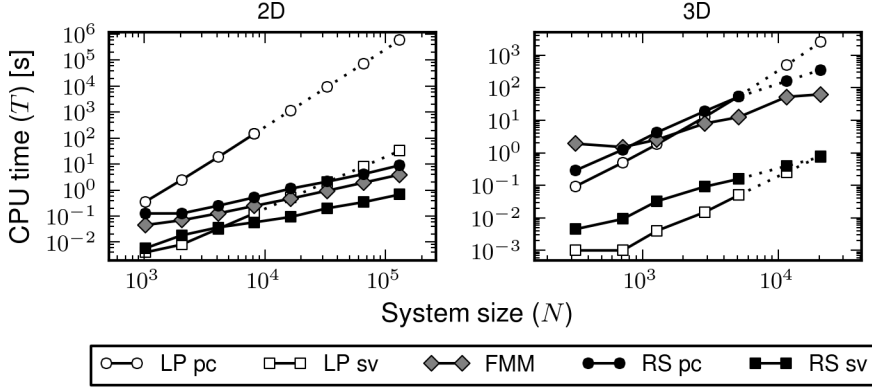


FIG. 7.1. CPU times for solving Laplace's equation in 2D and 3D using LAPACK/ATLAS (LP), FMM/GMRES (FMM), and recursive skeletonization (RS) as a function of the system size. For LP and RS, the computation is split into two parts: precomputation (pc), for LP consisting of matrix formation and factorization, and for RS of matrix compression and factorization; and system solution (sv), consisting of matrix (pseudo-) inverse application via precomputed QR factors. The precision of FMM and RS was set at $\epsilon = 10^{-9}$ in 2D and 10^{-6} in 3D. Dotted lines indicate extrapolated data; for RS in 3D, only factorization and inversion (as executed through MATLAB) are extrapolated.

where

$$G(r) = \begin{cases} -1/(2\pi) \log r & \text{if } d = 2, \\ 1/(4\pi r) & \text{if } d = 3 \end{cases}$$

is the free-space Green's function, $\nu_{\vec{y}}$ is the unit outer normal at $\vec{y} \in \partial\Omega$, and σ is an unknown surface density. Letting \vec{x} approach the boundary, standard results from potential theory [32] yield the second-kind Fredholm boundary integral equation

$$(7.1) \quad -\frac{1}{2}\sigma(\vec{x}) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\vec{y}}}(\|\vec{x} - \vec{y}\|) \sigma(\vec{y}) d\vec{y} = f(\vec{x})$$

for σ , assuming that $\partial\Omega$ is smooth. This is not a least squares problem, but it allows us to compare the performance of the sparse QR approach with our previous sparse LU results [38]. Of course, since the system (7.1) is square, the corresponding sparse embedding (2.4) can be solved without iteration.

In 2D, we took as the problem geometry a 2:1 ellipse, discretized via the trapezoidal rule, while in 3D, we used the unit sphere, discretized as a collection of flat triangles with piecewise constant densities. We also compared our algorithm against an FMM-accelerated GMRES solver driven by the open-source FMMLIB software package [25], which is a fairly efficient implementation (but not optimized using the plane wave representations of [29]). Timing results for each case are shown in Fig. 7.1, with detailed data for the recursive skeletonization scheme given in Tables 7.1 and 7.2. The precision was set to $\epsilon = 10^{-9}$ in 2D and 10^{-6} in 3D.

It is evident that our method scales as predicted, with precomputation and solution complexities of $\mathcal{O}(N)$ in 2D ($d = 1$), and $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$, respectively, in 3D ($d = 2$). In 2D, both phases are very fast, easily beating the uncompressed LAPACK/ATLAS solver in both time ($\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$ for precomputation and solution, respectively) and memory, and coming quite close to the FMM/GMRES solver as well. The same is essentially true in 3D over the range of problem sizes tested,

TABLE 7.1

Numerical results for solving Laplace's equation in 2D at precision $\epsilon = 10^{-9}$: N , uncompressed matrix dimension; K_r , final row skeleton dimension; K_c , final column skeleton dimension; T_{cm} , matrix compression time (s); T_{qr} , sparse QR factorization time (s); T_{sv} , (pseudo-) inverse application time (s); E , L^2 relative error.

N	K_r	K_c	T_{cm}	T_{qr}	T_{sv}	E
1024	30	30	3.1E-2	9.4E-2	5.8E-3	1.1E-9
2048	29	30	6.5E-2	5.8E-2	1.8E-2	4.5E-9
4096	30	30	1.3E-1	1.2E-1	3.7E-2	1.5E-8
8192	30	31	2.6E-1	2.6E-1	5.7E-2	1.4E-8
16384	31	31	5.2E-1	6.0E-1	9.2E-2	1.7E-8
32768	30	30	1.1E+0	1.0E+0	2.0E-1	1.2E-8
65536	30	30	2.1E+0	2.0E+0	3.4E-1	1.6E-8
131072	29	29	4.1E+0	4.7E+0	6.8E-1	2.2E-8

TABLE 7.2

Numerical results for solving Laplace's equation 3D at precision $\epsilon = 10^{-6}$. Extrapolated values are given in parentheses; all other notation as in Table 7.1.

N	K_r	K_c	T_{cm}	T_{qr}	T_{sv}	E
320	320	320	2.3E-1	5.1E-2	4.5E-3	1.5E-10
720	628	669	1.1E+0	1.6E-1	9.3E-3	5.1E-07
1280	890	913	3.7E+0	5.0E-1	3.2E-2	1.0E-06
2880	1393	1400	1.7E+1	1.9E+0	9.1E-2	1.2E-06
5120	1886	1850	4.7E+1	6.0E+0	1.6E-1	2.2E-06
11520	2750	2754	1.4E+2	(1.9E+1)	(3.9E-1)	
20480	3592	3551	3.1E+2	(4.6E+1)	(7.4E-1)	

though it should be emphasized that FMM/GMRES has optimal $\mathcal{O}(N)$ complexity and so should prevail asymptotically. However, as observed previously [27, 38, 42], the solve time using recursive skeletonization following precomputation (comprising one application of \mathbf{Q}^* and one backsolve with \mathbf{R}) is much faster than an individual FMM/GMRES solve: e.g., in 2D at $N = 131072$, $T_{FMM} = 3.9$ s, while $T_{sv} = 0.7$ s. This is significantly slower when compared with our UMFPACK-based sparse LU solver [38] ($T_{sv} \sim 0.1$ s). The difference may be due, in part, both to a higher constant inherent in the QR approach and to the overhead from interfacing with MATLAB. Unfortunately, we were unable to perform the sparse QR factorizations in-core for the 3D case beyond $N \sim 10^4$; the corresponding data are extrapolated via the results of section 5.

7.2. Least squares fitting of thin plate splines. We next turned to an overdetermined problem involving 2D function interpolation using thin plate splines; see (1.3). More precisely, we sought to compute the coefficients a_j of the interpolant

$$g(\vec{x}) = \sum_{j=1}^N a_j \phi_{TPS}(\|\vec{x} - \vec{c}_j\|),$$

that best matches a given function

$$f(x, y) = \sin(4\pi x) + \cos(2\pi y) \sin(3\pi xy) \quad \text{for } \vec{x} \equiv (x, y)$$

in the least squares sense on some set of randomly chosen targets $\vec{x}_i \in [0, 1]^2$ for $i = 1, \dots, M$. The points \vec{c}_j for $j = 1, \dots, N$ denote the centers of the splines and lie on a uniform tensor product grid on $[0, 1]^2$. Since this is somewhat ill-conditioned, we

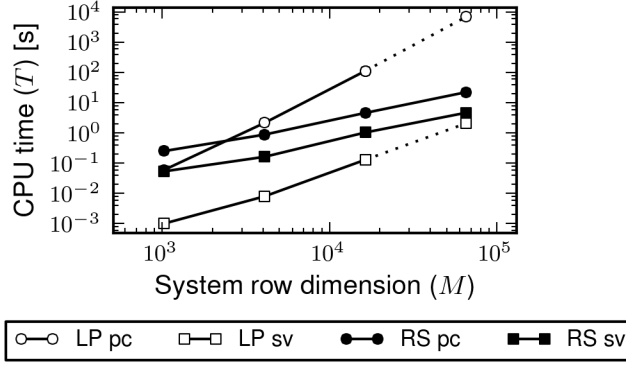


FIG. 7.2. CPU times for overdetermined thin plate spline fitting in 2D at precision $\epsilon = 10^{-6}$ using LAPACK/ATLAS and recursive skeletonization as a function of the system row dimension M , with the column dimension fixed proportionally at $N = M/4$; all other notation as in Fig. 7.1.

TABLE 7.3

Numerical results for overdetermined thin plate spline fitting in 2D at precision $\epsilon = 10^{-6}$: M , uncompressed row dimension; N , uncompressed column dimension; n_{iter} , number of deferred correction iterations; R , L^2 residual; all other notation as in Table 7.1.

M	N	K_r	K_c	T_{cm}	T_{qr}	T_{sv}	n_{iter}	E	R
1024	256	174	148	7.7E-2	1.7E-1	5.3E-2	1	4.1E-5	1.4E-1
4096	1024	260	247	5.7E-1	3.1E-1	1.6E-1	1	8.3E-5	4.4E-2
16384	4096	399	391	3.1E+0	1.5E+0	1.0E+0	1	3.9E-4	1.6E-2
65536	16384	564	574	1.5E+1	7.0E+0	4.7E+0	1	2.0E-6	6.7E-3

also add Tikhonov regularization with regularization parameter $\mu = 10^{-2}$ as indicated in (4.1).

Timing results for various M and N at a fixed ratio of $M/N = 4$ with $\epsilon = 10^{-6}$ are shown in Fig. 7.2, with detailed data in Table 7.3. The results are consistent with our complexity estimates of $\mathcal{O}(M + N^{3/2})$ and $\mathcal{O}(M + N \log N)$ for precomputation and solution, respectively. This compares favorably with the uncompressed complexities of $\mathcal{O}(MN^2)$ and $\mathcal{O}(MN)$, respectively, for LAPACK/ATLAS. Note also the convergence in the residual $\|f(\vec{x}) - g(\vec{x})\|$ on the targets \vec{x}_i of roughly second order. In all cases, the deferred correction procedure converged with just one iteration.

7.3. Underdetermined charge fitting. We then considered an underdetermined problem: seeking a minimum-norm charge distribution in 2D. The setup is as follows. Let \vec{x}_j for $j = 1, \dots, N$ be uniformly spaced points on the unit circle, each associated with a random charge q_j . We measure their induced field

$$f(\vec{x}; q) = -\frac{1}{2\pi} \sum_{j=1}^N q_j \log \|\vec{x} - \vec{x}_j\|$$

on a uniformly sampled outer ring of radius $1 + \delta$, and compute an equivalent set of charges \tilde{q}_j with minimal Euclidean norm reproducing those measurements. Here, we set $\delta = 10^{-4}$ and sampled at $M = N/8$ observation points over a range of N .

Timing results at precision $\epsilon = 10^{-9}$ are shown in Fig. 7.3, with detailed data given in Table 7.4. Since the source and target points are separated (by an annular region of width δ), our algorithm has optimal $\mathcal{O}(M + N)$ complexity, which is readily observed.

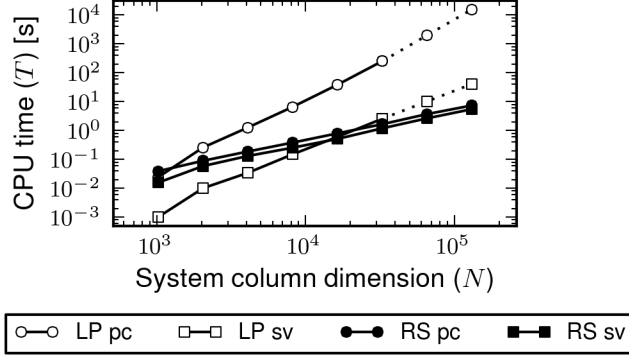


FIG. 7.3. CPU times for underdetermined charge fitting in 2D at precision $\epsilon = 10^{-9}$ using LAPACK/ATLAS and recursive skeletonization as a function of the system column dimension N , with the row dimension fixed proportionally at $M = N/8$; all other notation as in Fig. 7.1.

TABLE 7.4

Numerical results for underdetermined charge fitting in 2D at precision $\epsilon = 10^{-9}$; notation as in Table 7.3.

M	N	K_r	K_c	T_{cm}	T_{qr}	T_{sv}	n_{iter}	E	R
128	1024	69	72	1.7E-2	2.1E-2	1.6E-2	1	1.6E-9	1.6E-15
256	2048	80	80	4.4E-2	4.3E-2	5.8E-2	2	1.3E-8	1.1E-15
512	4096	89	90	9.6E-2	8.7E-2	1.3E-1	2	6.1E-8	1.9E-15
1024	8192	99	100	2.0E-1	1.8E-1	2.5E-1	2	5.5E-8	3.0E-15
2048	16384	108	110	4.0E-1	3.8E-1	5.1E-1	2	3.6E-8	1.8E-15
4096	32768	119	119	8.1E-1	8.2E-1	1.2E+0	2	3.5E-8	2.1E-15
8192	65536	128	131	1.6E+0	2.0E+0	2.6E+0	2	4.8E-9	7.1E-09
16384	131072	134	138	3.3E+0	4.0E+0	5.5E+0	2	6.6E-9	7.5E-09

Furthermore, as we have solved an approximate, compressed system, we cannot in general fit the data exactly. Indeed, we see residuals $\|f(\vec{x}; q) - f(\vec{x}; \tilde{q})\|$ of order $\mathcal{O}(\epsilon)$ as predicted by the compression tolerance. Thus, our algorithm is especially suitable in the event that observations need to be matched only to a specified precision. Deferred correction was successful in all cases within two iterations.

7.4. Thin plate splines with updating. In our final example, we demonstrate the efficiency of our updating and downdating methods in the typical setting of fitting additional observations to an already specified overdetermined system. For this, we employed the thin plate spline approximation problem of section 7.2 with $M = 16384$ and $N = 4096$, followed by the addition of 50 new, random target points. From section 6, the perturbed system can be written as (6.1) with (6.2), i.e.,

$$\mathbf{E}\mathbf{x} \simeq \mathbf{f} \quad \text{where } \mathbf{E} = \begin{bmatrix} A \\ C_+^r \end{bmatrix}.$$

We used GMRES with the left preconditioner $\mathbf{B} = [A^+, (C_+^r)^*]$, which, since A has full column rank, gives $\mathbf{B}\mathbf{E} = I + (C_+^r)^* C_+^r$, hence the preconditioned system is

$$[I + (C_+^r)^* C_+^r] \mathbf{x} \simeq \mathbf{B}\mathbf{f}.$$

Note that only *one* application of A^+ is necessary, independent of the number of iterations required. Solving this in MATLAB took 17 iterations and a total of 1.9

s, with 1.6 s going towards setup. The residual on the new data was 7.7×10^{-3} . This should be compared with the roughly 6 s required to solve the problem without updating, treating it instead as a new system via our compressed algorithm (Table 7.3). Although this difference is perhaps not very dramatic, it is worth emphasizing that the complexity here scales as $\mathcal{O}(M + N \log N)$ with updating versus $\mathcal{O}(M + N^{3/2})$ without, as the former needs only to apply A^+ while the latter needs also to compress and factor A . Therefore, the asymptotics for updating are much improved.

8. Generalizations and conclusions. In this paper, we have presented a fast direct algorithm for overdetermined and underdetermined least squares problems involving HBS matrices, and exhibited its efficiency and practical performance in a variety of situations including RBF interpolation and dynamic updating. In 1D (including boundary problems in 2D and problems with separated data in all dimensions), the solver achieves optimal $\mathcal{O}(M + N)$ complexity and is extremely fast, but it falters somewhat in higher dimensions, due primarily to the growing ranks of the compressed matrices as expressed by (5.1). Developments for addressing this growth are now underway for square linear systems [18], and we expect these ideas to carry over to the present setting. Significantly, the term involving the larger matrix dimension is linear in all complexities (i.e., only $\mathcal{O}(M)$ instead of $\mathcal{O}(MN^2)$ as for classical direct methods), which makes our algorithm ideally suited for large, rectangular systems where both M and N increase with refinement.

Although we have not explicitly considered least squares problems with HBS equality constraints (we have only done so implicitly through our treatment of underdetermined systems), it is evident that our methods generalize. However, our complexity estimates can depend on the structure of the system matrix. In particular, if it is sparse, e.g., a diagonal weighting matrix, then our estimates are preserved. We can also, in principle, handle HBS least squares problems with HBS constraints simply by expanding out both matrices in sparse form.

Finally, it is worth noting that fast direct solvers can be leveraged for other least squares techniques as well. This is straightforward for the normal equations, which are subject to well-known conditioning issues, and for the somewhat better behaved augmented system version [2, 8]:

$$\begin{bmatrix} I & A \\ A^* & \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

This approach has the advantage of being immediately amenable to fast inversion techniques, but it symmetrizes the system. Thus, all complexity estimates involve $M + N$ instead of M and N separately. In particular, the current generation of fast direct solvers would require, e.g., $\mathcal{O}((M + N)^{3(1-1/d)})$ instead of $\mathcal{O}(M + N^{3(1-1/d)})$ work. With the development of a next generation of linear or nearly linear time solvers [18, 34], this distinction may become less critical. Memory usage and high-performance computing hardware issues will also play important roles in determining which methods are most competitive. We expect these issues to become settled in the near future.

REFERENCES

- [1] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSSEN, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, 3rd ed., 1999.

- [2] M. ARIOLI, I. S. DUFF, AND P. P. M. DE RIJK, *On the augmented system approach to sparse least-squares problems*, Numer. Math., 55 (1989), pp. 667–684.
- [3] J. L. BARLOW, *Error analysis and implementation aspects of deferred correction for equality constrained least squares problems*, SIAM J. Numer. Anal., 25 (1988), pp. 1340–1358.
- [4] J. L. BARLOW AND S. L. HANDY, *The direct solution of weighted and equality constrained least-squares problems*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 704–716.
- [5] J. L. BARLOW AND U. B. VEMULAPATI, *A note on deferred correction for equality constrained least squares problems*, SIAM J. Numer. Anal., 29 (1992), pp. 249–256.
- [6] C. I. BAYLY, P. CIEPLAK, W. D. CORNELL, P. A. KOLLMAN, *A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: the RESP model*, J. Phys. Chem., 97 (1993), pp. 10269–10280.
- [7] P. BENNER AND T. MACH, *On the QR decomposition of \mathcal{H} -matrices*, Computing, 88 (2010), pp. 111–129.
- [8] A. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA, 1996.
- [9] S. BÖRM, *Data-sparse approximation of non-local operators by \mathcal{H}^2 -matrices*, Linear Algebra Appl., 422 (2007), pp. 380–403.
- [10] M. D. BUHMANN, *Radial Basis Functions: Theory and Implementation*, Cambridge University Press, Cambridge, 2003.
- [11] F. BURNS, D. CARLSON, E. HAYNSWORTH, AND T. MARKHAM, *Generalized inverse formulas using the Schur complement*, SIAM J. Appl. Math., 26 (1974), pp. 254–259.
- [12] S. L. CAMPBELL AND C. D. MEYER JR., *Generalized Inverses of Linear Transformations*, Pitman, London, 1979.
- [13] J. C. CARR, R. K. BEATSON, J. B. CHERRIE, T. J. MITCHELL, W. R. FRIGHT, B. C. MCCALLUM, AND T. R. EVANS, *Reconstruction and representation of 3D objects with radial basis functions*, in Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, 2001, pp. 67–76.
- [14] S. CHANDRASEKARAN, P. DEWILDE, M. GU, T. PALS, X. SUN, A.-J. VAN DER VEEN, AND D. WHITE, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.
- [15] S. CHANDRASEKARAN, P. DEWILDE, M. GU, W. LYONS, AND T. PALS, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.
- [16] S. CHANDRASEKARAN, M. GU, AND T. PALS, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
- [17] H. CHENG, Z. GIMBUTAS, P.-G. MARTINSSON, AND V. ROKHLIN, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404.
- [18] E. CORONA, P.-G. MARTINSSON, AND D. ZORIN, *An $O(N)$ direct solver for integral equations on the plane*, preprint.
- [19] T. A. DAVIS, *Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Softw., 30 (2004), pp. 196–199.
- [20] T. A. DAVIS, *Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization*, ACM Trans. Math. Softw., 38 (2011), pp. 8:1–8:22.
- [21] P. DEWILDE AND S. CHANDRASEKARAN, *A hierarchical semi-separable Moore-Penrose equation solver*, Oper. Theor. Adv. Appl., 167 (2006), pp. 69–85.
- [22] J. DUCHON, *Splines minimizing rotation-invariant semi-norms in Sobolev spaces*, in Constructive Theory of Functions of Several Variables, Lecture Notes in Mathematics, 571, W. Schempp and K. Zeller, eds., Springer-Verlag, Berlin, 1977, pp. 85–100.
- [23] M. M. FRANCL, C. CAREY, L. E. CHIRLIAN, AND D. M. GANGE, *Charges fit to electrostatic potentials. II. Can atomic charges be unambiguously fit to electrostatic potentials?*, J. Comput. Chem., 17 (1996), pp. 367–383.
- [24] A. GILLMAN, P. M. YOUNG, AND P.-G. MARTINSSON, *A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains*, Front. Math. China., 7 (2012), pp. 217–247.
- [25] Z. GIMBUTAS AND L. GREENGARD, *FMMLIB: fast multipole methods for electrostatics, elastostatics, and low frequency acoustic modeling*, in preparation. Software available from <http://www.cims.nyu.edu/cmcl/software.html>.
- [26] L. GRASEDYCK, *Hierarchical singular value decomposition of tensors*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2029–2054.
- [27] L. GREENGARD, D. GUEYFFIER, P.-G. MARTINSSON, AND V. ROKHLIN, *Fast direct solvers for integral equations in complex three-dimensional domains*, Acta Numer., 18 (2009), pp. 243–275.
- [28] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

- [29] L. GREENGARD AND V. ROKHLIN, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., 6 (1997), pp. 229–269.
- [30] T. N. E. GREVILLE, *Some applications of the pseudoinverse of a matrix*, SIAM Rev., 2 (1960), pp. 15–22.
- [31] M. GU, *New fast algorithms for structured linear least squares problems*, SIAM J. Matrix Anal. Appl., 20 (1998), pp. 244–269.
- [32] R. B. GUENTHER AND J. W. LEE, *Partial Differential Equations of Mathematical Physics and Integral Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
- [33] W. HACKBUSCH, *A sparse matrix arithmetic based on \mathcal{H} -matrices. Part I: Introduction to \mathcal{H} -matrices*, Computing, 62 (1999), pp. 89–108.
- [34] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive \mathcal{H}^2 -matrices*, Computing, 69 (2002), pp. 1–35.
- [35] W. HACKBUSCH AND B. N. KHOROMSKIJ, *A sparse \mathcal{H} -matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.
- [36] R. L. HARDY, *Multiquadric equations of topography and other irregular surfaces*, J. Geophys. Res., 76 (1971), pp. 1905–1915.
- [37] K. HAYAMI, J.-F. YIN, AND T. ITO, *GMRES methods for least squares problems*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2400–2430.
- [38] K. L. HO AND L. GREENGARD, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–A2532.
- [39] T. KAILATH AND A. H. SAYED, *Fast Reliable Algorithms for Matrices with Structure*, SIAM, Philadelphia, PA, 1999.
- [40] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
- [41] E. LIBERTY, F. WOOLFE, P.-G. MARTINSSON, V. ROKHLIN, AND M. TYGERT, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.
- [42] P.-G. MARTINSSON AND V. ROKHLIN, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
- [43] C. C. PAIGE AND M. A. SAUNDERS, *LSQR: an algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw., 8 (1982), pp. 43–71.
- [44] M. J. D. POWELL, *Radial basis functions for multivariable interpolation: a review*, in Algorithms for Approximation, J. C. Mason and M. G. Cox, eds., Clarendon Press, Oxford, 1987, pp. 143–167.
- [45] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
- [46] H. SAMET, *The quadtree and related hierarchical data structures*, ACM Comput. Surv., 16 (1984), pp. 187–260.
- [47] M. VAN BAREL, G. HEINIG, AND P. KRAVANJA, *A superfast method for solving Toeplitz linear least squares problems*, Linear Algebra Appl., 366 (2003), pp. 441–457.
- [48] C. VAN LOAN, *On the method of weighting for equality-constrained least-squares problems*, SIAM J. Numer. Anal., 22 (1985), pp. 851–864.
- [49] R. C. WHALEY, A. PETITET, AND J. J. DONGARRA, *Automated empirical optimization of software and the ATLAS project*, Parallel Comput., 27 (2001), pp. 3–35.
- [50] F. WOOLFE, E. LIBERTY, V. ROKHLIN, AND M. TYGERT, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.
- [51] J. XIA, S. CHANDRASEKARAN, M. GU, X. S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
- [52] J. XIA, Y. XI, AND M. GU, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.
- [53] L. YING, G. BIROS, AND D. ZORIN, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626.